미들웨어와 UML을 활용한 교환 소프트웨어의 개발과 관리

Managing and Development of Switching Software Using Middleware and UML

이재기(J.K. Lee) 시스템종합팀 선임연구원
신상권(S.K. Shin) 시스템종합팀 연구원
이수종(S.J. Lee) 시스템종합팀 책임연구원
남상식(S.S. Nam) 시스템종합팀 책임연구원, 팀장

본 논문은 대형 ATM 스위칭 시스템인 HANbit ACE2000 시스템의 소프트웨어를 개발하고 시스템 개발 시 필요한 각종 소프트웨어 컴포넌트들을 효율적으로 개발 및 등록, 관리하기 위해 도입된 미들웨어와 객체지향 설계 개념을 이용한 UML을 적용하여 프로그램의 에러를 줄이고 개발자와 관리자간의 규약에 의한 체계적인 종합개발 및 관리 방안에 대한 특성과 각 기능에 대해 살펴본다. 또 이것을 활용하여 ACE2000 시스템에서 개발, 등록 관리되고 있는 소프트웨어에 대해 소개함으로써 대형 시스템 개발에 유용하게 이용되도록 한다. 특히, 대형 시스템 개발에 적용된 개발체계 및 개발환경을 언급함으로써 유사 프로젝트 수행 시 추가 개선사항이나 개발환경 개선에 참고가 되도록 하였다.

I. 서 론

소프트웨어에 대한 사용자 요구사항이 다양해지고 복잡해짐에 따라 이에 대한 적기 대응을 위한 다양한 개발방법론이 대두되었다[1]. 특히 대규모 교환 소프트웨어를 개발하는 데 있어서도 예외가 아니었으며, 기존의 개발방법을 과감히 배제하고 효율적이고 체계적으로 소프트웨어를 개발하려고 도입한 개념이 소프트웨어 컴포넌트 기반의 소프트웨어 개발 방법론이다[2].

ACE2000 시스템에서는 이를 위해서 미들웨어를 채용하여 응용프로그램을 개발함으로써 개발기간의 단축 및 생산성을 향상시키고, 또 소프트웨어의 개발과정과 전체적인 흐름을 이해시켜 소프트웨어 개발의 체계와 효율을 극대화하였다.

컴포넌트 기술의 성숙에 따른 소프트웨어 재사용이 가능해짐에 따라 개발비용의 절감 및 개발기간의 단축, 생산성의 향상을 가져왔으나 한편으로는 각 컴포넌트가 구현하는 인터페이스의 구문적인 측면을 기술하는 IDL(Interface Definition Language)이 필요하게 된다. 즉 IDL은 컴포넌트의 기능을 사용하기 위한 클라이언트의 구문적인 적합성만 보장하기 때문에 컴포넌트의 올바른 사용을 보장받기 위해서는 컴포넌트 명세에 의미적인 측면이 기술되어야 한다[3].

컴포넌트에 대한 일반적인 명세방법은 IDL과 사용자 매뉴얼, 데모 프로그램 예시 정도이다. 이렇기 때문에 컴포넌트에 대한 이해에 많은 어려움이 따른다. 이러한 이유로 경험 부족과 툴의 이해 부족에 따른 여파로 시스템 개발 초기에 많은 에러를 유발하

였다. 그래서 이것과 별도로 도입한 개념이 객체지향 프로그래밍 기법으로 시스템의 기능 중 구현이 복잡하고 에러 유발이 심한 운용보전 기능에 대해서 Rational사의 UML(Unified Modeling Language) 툴을 도입하여 프로그램을 개발하고 또 미들웨어와 상호 보완적인 기법을 도입하여 소프트웨어의 생산성 및 품질을 높였다.

본 논문의 구성은 Ⅱ장에서 미들웨어를 이용한 프로그램 개발방법에 대해 살펴보고 Ⅲ장에서는 UML을 활용한 예를, Ⅳ장에서는 Ⅱ, Ⅲ장에서 소개한 미들웨어와 UML을 활용하여 개발한 교환 소프트웨어의 실행결과와 패키지 종합환경에서 관리하고 있는 각종 소프트웨어 컴포넌트 관리 현황에 대해 살펴본다. 그리고 마지막에 향후 연구방향 및 결론을 맺는다.

Ⅱ. 미들웨어를 이용한 응용 프로그램 개발

ACE2000 시스템에서는 분산 환경에서 이기종(異機種) 간의 서버와 클라이언트들을 연결해주는 소프트웨어로 서로 다른 운영체제(OS)와 서버 프로그램과의 호환성 및 이종의 통신프로토콜을 사용하는 네트워크간의 접속, 그리고 시스템을 연결해 단일 사용자 환경으로 만들어주는 미들웨어를 채택하였다.

미들웨어(Netfree 2000)에 기반한 응용 프로그램의 개발은 특별한 환경의 제공이 필요한데, 이 환경은 개발 대상 시스템에 따라 대응하는 환경을 갖춤으로써 프로그램 개발을 용이하게 수행할 수 있다[4].

이에 대한 세부 개발환경에 대한 사항은 <표 1>과 같다.

<표 1> 개발환경 예시

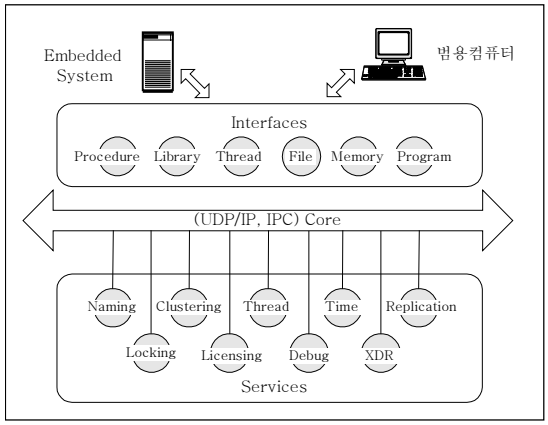
구분	내용
운영체제	SUN/LINUX
네트워크 프로토콜	UDP/IP
개발언어	C++
컴파일러	G++(GNU compiler)

그밖에 사전 준비사항으로는 Java VM(Virtual Machine), 시스템 구성(Name Table) 등록 및 NS(Name Server)의 실행, UI(User Interface로서 netfreed[netfree daemon]를 의미함)의 실행 등이다.

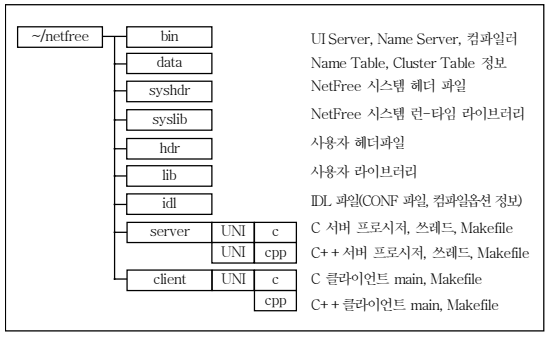
1. 개발환경 및 구성

미들웨어 개발환경은 개발자 및 각 컴포넌트들을 등록하고 개발자의 계정을 관리하며 시스템 구성 및 클러스터링 정보를 등록 관리하는 관리자로 구분된다. 대부분의 개발작업은 UI를 통해 수행되며, 등록 관리되고 있는 서비스 구성도 및 디렉토리 구조는 (그림 1), (그림 2)와 같다.

서비스 처리에 대한 각 부분별 기능은 다음과 같다.



(그림 1) Netfree 2000 서비스 구조



(그림 2) Netfree 2000 디렉토리 구조

가. Naming(명명 서비스)

서비스 개념은 시스템 내부에 존재하는 자원에 대해 논리적 이름을 부여하고 해당 이름을 통해 물리적인 주소를 반환하여 외부에서 접근이 가능토록 하는 서비스이다.

나. Clustering

복수의 컴퓨터를 연결하여 하나의 시스템처럼 작동하도록 하며, 서버 다운 시 대체하여 연속작업이 가능토록 한다.

다. Thread

미들웨어를 이용한 서버가 복수개의 thread를 처리하도록 한다. 즉, 클라이언트로부터 요청한 job을 신속히 처리

라. Time-out

임의 서버가 서비스 요청 시 서버의 비정상 동작이나 네트워크 문제로 인해 서비스가 불가능한 경우 오류 발생을 알리고 다른 처리를 하도록 한다.

마. Replication(중복)

여러 서버에 대해 해당 서비스가 동시에 처리되도록 하는 서비스이다.

바. Locking

공유 자원을 복수개의 프로그램이 사용하는 경우 데이터의 무결성을 보장하기 위한 방법으로 net-free에서 서버 및 파일, 메모리, 초기화 등에 lock을 제공한다.

사. License

공급자와 사용자간의 계약사항을 유지한다.

아. Debug

작성된 프로그램에서 발생되는 문제점 해결을 위한 지원기능으로 trace 및 입·출력 제어 등을 수행한다(즉, 출력 레벨을 임의로 조정하여 원하는 레벨의 로그만을 출력하도록 한다).

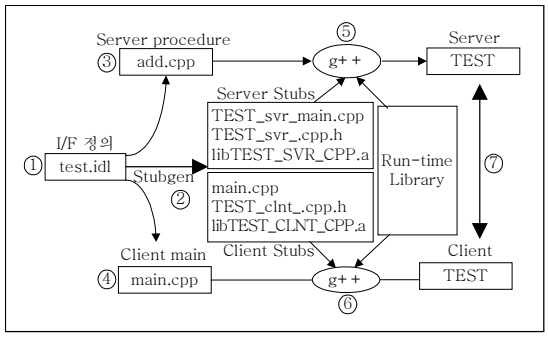
자. XDR(External Data Representation)

장치간의 차이를 극복하기 위한 송·수신 데이터의 내부 변환을 담당한다.

2. 프로그램 개발 순서

미들웨어를 활용한 프로그램 개발순서는 아래와 같으며, 그 과정은(그림 3)과 같다.

① 문제의 정의
② UI 클라이언트 수행
③ IDL 작성 및 컴파일
④ 서버 프로시저 작성
⑤ 클라이언트 프로그램 작성
⑥ 서버 프로그램 컴파일 및 수행
⑦ 클라이언트 프로그램 컴파일 및 수행



(그림 3) 개발과정

문제의 정의(혹은 UI)는 개발자가 set, get procedure와 해당 procedure를 호출하는 클라이언트, 클라이언트가 프로시저를 원격으로 호출할 수 있도록 인터페이스를 정의하는 작업을 해주어야 한다.

IDL 컴파일을 수행하면 이때 생성된 결과들(*.h: header, library & *.idl)은 패키지 종합절차에 따른 해당 관리 디렉토리에 정보들이 자동으로 이동,

저장되어 패키지 제작절차에 따라 각 단계별로 수행된다(세부 사항은 부록의 디렉토리 계층구조 참조).

3. 서버의 구성

미들웨어 서버(netfree server)는 크게 논리적인 서버(logical server)와 물리적인 서버(physical server)로 분류된다.

논리적인 서버는 분산환경에서 물리적 위치에 관계없이 논리적인 이름(name)을 통해 접근이 가능하게 되는데 이때 미들웨어 서버는 여러 노드(node)-워크스테이션(W/S) 또는 Embedded-system에 존재할 수 있다.

각 서버들은 다른 프로그램에 대해 독립적으로 실행되지만 라이브러리 기능을 수행하도록 작성된 서버는 다른 프로그램에 대해서는 명시적으로 지정되어야 한다. 한편 물리적인 서버는 고유한 주소를 갖는 특정 노드에서 수행된다.

응용 프로그램 개발과정에서의 논리적인 서버는 시스템의 구성에 따라 물리적인 서버를 인식할 수 있는데 그 구성 예는 <표 2>와 같이 나타낼 수 있다.

<표 2> 미들웨어 서버의 구성

• 논리적인 관점에서의 서버

		서버						
기능서버	서버명	UNI			NNI		WS	
동작서버	일련번호	0	1	2	0	1	0	1

• 물리적인 관점에서의 서버

서버명	서버 ID	일련번호	IP 주소	IPC 주소
UNI	0	0	129.254.131.34	0x1f80
UNI	0	1	129.254.131.35	0x0000
UNI	0	2	129.254.131.36	0x0080
NNI	1	0	129.254.131.37	0x0180
NNI	1	1	129.254.131.38	0x0280
WS	2	0	129.254.131.32	0x1e00
WS	2	1	129.254.131.33	0x1f00

가. NS의 기능

NS는 기능 서버들을 등록, 관리하며 등록된 서버에 대한 클라이언트 정보요청에 대해 해당 정보를 제공해주는 역할을 한다.

NS가 제공하는 주요 기능으로는

① 기능서버의 인증 및 등록기능

② 등록 서버의 주소정보 제공

③ 동일 이름의 서버 인스턴스 개수 제공

④ 기능서버가 사용하는 프로토콜 제공

⑤ 상·하위 NS와 기능 서버들의 상태관리

등이 있다.

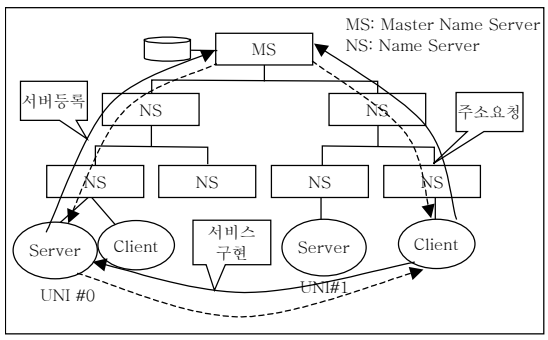
나. NS의 구조 및 환경

NS는 계층적 트리구조로 MS(Master Server)는 운용시스템에서 수행되는 NS로 UI에서 등록한 시스템 구성정보를 읽어 들여 보관한다. 또 하위에 여러 NS를 두며 NS의 정보 요청에 대해 서비스를 해준다.

중간 계층의 NS에 대한 NS Table 정보는 상위 NS로부터 받고 NS 정보를 직접 읽지 않는 것을 제외하고는 MS와 동일한 기능을 수행한다(<표 3>, (그림 4) 참조).

<표 3> NS 환경

프로토콜	UDP	IPC	UDP & IPC
사용 OS	SROS MSROS UNIX LINUX	UNIX(SROSD) SROS, MSROS	UNIX(SROSD) SROS, MSROS
사용포트	9000		9000
시그널 ID		0XF805, 0XF831	0XF805, 0XF831



(그림 4) NS의 구조

다. NS 시동 절차

NS는 계층적 구조로 하위 NS가 먼저 수행되어도 상위 계층의 NS가 수행될 때까지 서비스가 제공되지 않고 대기중으로 지속한다. 즉, ACE2000 시스템에서는 워크스테이션 측에 올라가는 master NS인 MS(워크스테이션이 이중화로 동작중인 경우에는 master/slave에 의한 MSM/MSS로 구분함)와 시스템의 메인 프로세서에 올라가는 NS로 구분되고 NS가 먼저 로딩되어 run 되어도 워크스테이션 측의 MS가 run 되지 못한 상태에서는 정상적인 통신이 되지 않는다.

우선 워크스테이션과 시스템간의 UDP 통신을 정상적으로 수행하기 위해서는 워크스테이션 측의 net-free table에 NS가 등록되어야 한다(<표 4> 참조).

<표 4> Name Table 구성 예

000 NS 0 0 –1 0 129.254.131.34 9000 e000 0 f805 0 –1 –1 0
000 NS 1 0 –1 0 129.254.131.34 9000 f000 0 f805 0 –1 –1 0
000 NS 2 0 –1 0 129.254.131.34 9000 f002 0 f805 0 –1 –1 0

* E000: 워크스테이션, F000: ACC 0, F002: SMC

◈ NS 수행 절차 및 에러 유형

① MS를 수행하여 Name table에 등록된 정보와 NS 정보를 읽어 들인다.
② NS를 수행하면 NSM으로부터 필요한 정보를 전달받는다.
③ 기타 에러 발생에 따른 조치(can't bind local address, name table not exist, [MHRPC] SERVER REGISTED FAIL etc.)
④ 동작상태 확인(<표 5> 참조)

NS의 서버 출력상태가 NORM이면 정상적으로 동작하고 있는 상태이며, FAIL인 경우는 해당 블록 또는 FS(File System), Server[NS x]가 비정상 상태로 이때는 해당 블록의 상태를 시스템의 프로세서 또는 워크스테이션의 데몬 프로세스(daemon process) 상태를 확인한 후 정상상태로 만들어 정상적인 통신이 이루어지도록 조치한다.

워크스테이션에서 MSM(Master Nameserver)을 구동하였을 때 앞에 언급한 NS 수행절차의 ③항의 에러가 자주 발생하는데([MHRPC] SERVER REGISTED FAIL) 메시지가 발생할 때는 워크스테이션의 이중화 처리 editDM 파일에 해당 블록이 등록되어 있는지 또는 자동실행이 되도록 설정되어 있는가를 확인한다.

이 경우는 워크스테이션의 netfree 테이블(file 위치: /atm/data/netfreetab, 워크스테이션 별로 어드레스를 구분함: netfreetab.200(205) 등의 파일이 존재함)의 주소 정보를 확인하여 시스템과 워크스테이션 간의 IP 정보 및 Name table 정보가 일치하는지를 검사한다. 이 정보가 맞지 않거나 이미 워크스테이션에 NS(MSM/MSS)가 동작중에 다시 수행시키면 에러로 출력한다. 이때는 해당 데이터를 수정하고 NS table(netfreetab) 파일이 위치한 디렉토리에서 로그파일(netfreelog)을 삭제한 후 MSM을 재시동하면 된다. 그리고 워크스테이션의 데몬 프로세스 상태는 /atm/daemon/*(각 프로세스로 예를 들면 WSLSF, WSCFGDF, WSSFMF 등이 존재)를 확인한다. 파일 구동상태의 확인 방법은 /atm 산하 디렉토리에서 데몬 프로세스 상태를 확인하는 "dmsts"라는 명령어를 사용하여 동작상태를 확인한다.

III. UML을 활용한 소프트웨어 개발 및 관리

교환기 소프트웨어 중 시스템의 상태관리나 운용, 유지보수를 위한 프로그램 개발은 시스템의 안정화와 직결되는 문제로 매우 중요하게 여겨 왔으며, 프로그램의 양이나 복잡성이 높기 때문에 많은 결함을 유발한다. 이러한 결함을 줄이고 양질의 프로그램 개발을 위해 객체지향(object-oriented) 설계개념을 도입하고 UML을 적용하였다.

이외에 개발된 소프트웨어 관리와 프로그램 개발을 용이하게 하기 위한 패키지 관리 체계는 UML을 이용하였다. 즉, 다수의 개발자가 설계, 제작한 클래스를 공유하면서 전체 시스템의 모델링을 효율적으

<표 5> Master NS 출력 메시지 예

STATUS	SVRID	SEQ	NAME	PNT	IPADDR	PORT	IPC	SIG	PID	PROTO	MULTI	CLUST	VER
FAIL	0	0	NS0	-1	129.254.131.128	9000	e000	f805	0	UDP	SINGLE	255	N/A
NORM	2	0	NS2	0	129.254.131.130	9000	f000	f805	55ba	UDP	SINGLE	255	N/A
NORM	10000	2	FS	2	129.254.131.130	4747	f000	f831	0	UDP	SINGLE	255	N/A
NORM	10001	0	CDMF	2	129.254.131.130	4756	f000	f831	0	UDP	SINGLE	255	N/A
NORM	10002	0	DKMGF	2	129.254.131.130	4754	f000	f831	0	UDP	SINGLE	255	N/A
NORM	10003	0	S_IMMF_ACC	2	129.254.131.130	4739	f000	f831	0	UDP	SINGLE	255	N/A
NORM	10004	0	NADHF	2	129.254.131.130	4751	f000	f831	0	UDP	SINGLE	255	N/A

FS Warning FD_FULL in fd_get()

[U] 104. [STS_CHK]	16, recv: 129.254.131.128	9000
[U] 104. [STS_CHK]	16, send: 129.254.131.128	9000
[U] 95. [FS_STSCHK]	16, recv: 129.254.131.130	16384
[U] 95. [FS_STSCHK]	16, recv: 129.254.131.130	4753
[U] 95. [FS_STSCHK]	16, recv: 129.254.131.130	4754

* [U]: UDP, 104[95]: service ID, 16: data size, 9000[4754, …]: port ID, FS: file system

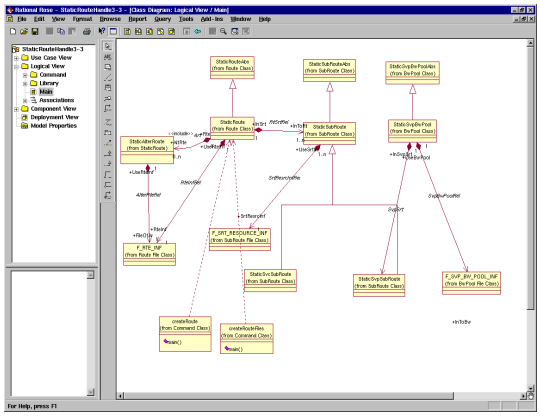
로 분산, 종합할 수 있는 체계가 요구되며 상호간 참조되는 모델을 이용하여 C 또는 C++ 코드를 해당 디렉토리에 적합하게 생성시켜야 한다. 이렇게 생성된 코드는 개발자와 관리자간의 상호 규약에 의해 해당 디렉토리에 생성, 등록 관리되어야 한다[5].

1. 코드 생성을 위한 시스템 구성

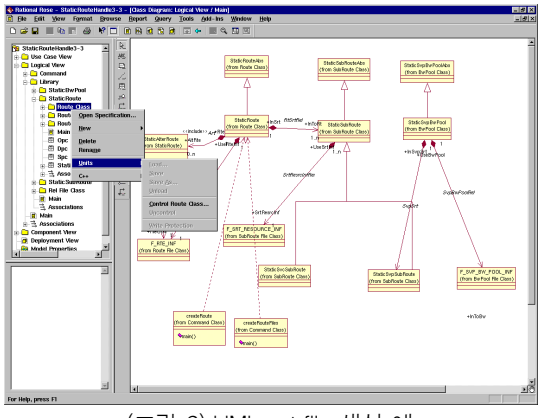
코드를 작성하는 데 있어서 확정된 디렉토리 구조에 적합하게 코드를 UML을 통해 생성하기 위해서는 Component Package 기능을 잘 활용하여야 한다. 또 mdl에서 모델링된 클래스를 타 기능 및 타인에게 제공할 필요가 있는 경우 클래스에 대한 cat 파일을 작성하고 Component Package에 대해서는 sub 파일을 작성하여 타 mdl에서 참조가 가능하도록 한다.

(그림 5)는 코드 생성을 위한 UML 샘플 구성이다. Logical view에서 각각의 클래스 기능에 따라 패키지를 생성, 소속시키고 이 패키지들은 다른 mdl에서 클래스를 참조시키기 위한 cat file 구성에 이용된다.

Logical view에서 cat file 생성을 위한 목적은 다른 mdl file 내의 클래스들에게 제공하는 인터페이스 역할과 mdl file 내 클래스에서 원하는 관계설정을 제공한다. 이에 대한 생성방법은 (그림 6)과 같다.



(그림 5) 코드 생성 이해를 위한 UML 샘플



(그림 6) UML cat file 생성 예

UML 코드 생성을 위한 일련의 절차는 아래와 같다.

① 프로젝트 -- Component Package -- Component – calss 의 계층적 관계를 이해하여 코드 생성 디렉토리를 확정
② cat file & sub file 작성 – 타 mdl과 상호 연관 관계 이해 등

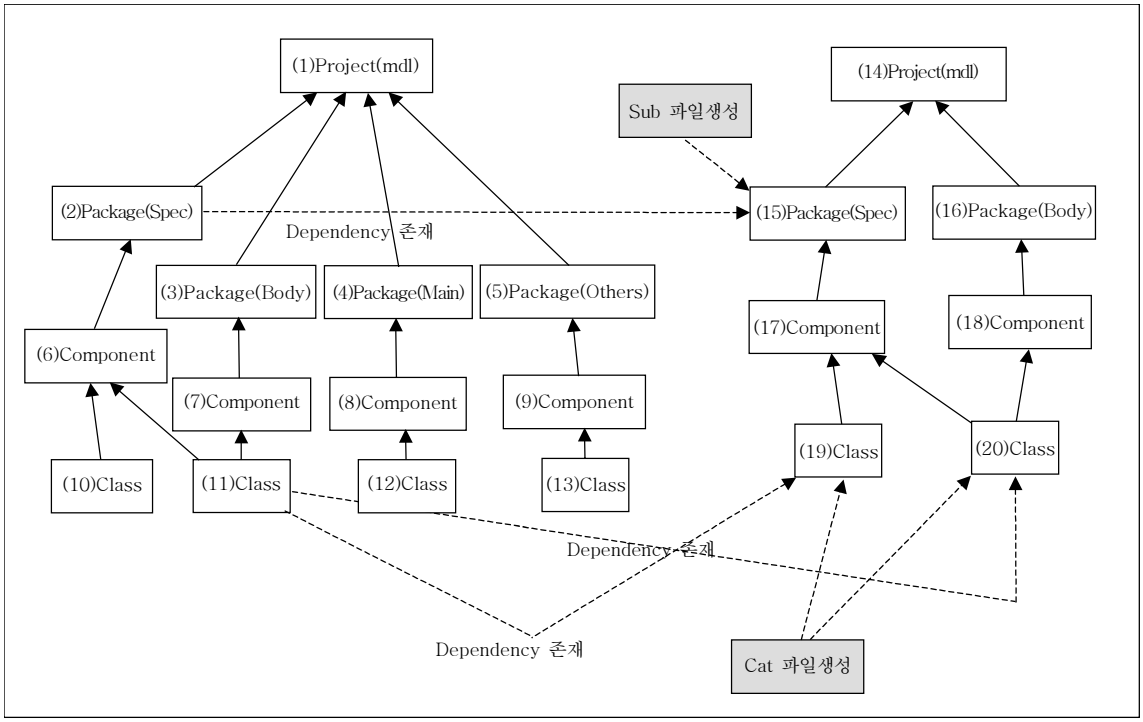
그밖에 아래의 사항이 부수적으로 수반된다.
③ C++을 사용하는 경우에는 UML로 소스를 생성하여 코딩
④ 호 제어 기능과 같이 사전에 UML 사용을 고려하지 못한 경우에는 UML을 사용하지 않음
⑤ UML은 블록 단위의 모든 소스에 적용
⑥ 블록 단위의 모든 소스에 사용하기 어려운 경우에는 적용 가능한 부분을 라이브러리 블록으로 구성 등

코드 생성을 위한 전체 시스템의 구성도를 살펴보면 (그림 7)과 같다.

2. 생성되는 코드와 개념

UML에서 제공되는 코드들은 클래스 헤더 디렉토리(clshdr)의 *.h 나 기능블록(function block) 밑에 저장되는 *.cpp(혹은 .ec) 파일들이며, (그림 5)에서 살펴보면 클래스가 모여서 컴포넌트를 구성하며 이때 생성되는 파일이 .h 와 .cpp 이다. 옵션으로 *.ec file로 생성되는 경우가 있는데 이것은 DB 접근 클래스로 이 파일들은 나중에 *.cpp file로 이동시켜 생성한다.

각 컴포넌트가 모여서 컴포넌트 패키지(혹은 subsystem)를 구성하며 이것은 파일의 묶음으로 ACE2000 시스템의 기능블록이나 블록의 헤더에 해당된다. 이 subsystem이 모여서 하나의 프로젝트를 구성하며 이 프로젝트는 단일 기능으로 각각의 컴파일 단위가 필요하고 이것들이 모여서 하나의 기능이 형성된다. 즉, 여러 프로젝트가 모여서 시스템을 구성하는데 이것은 서로 다른 기능들이 연동(interworking)하여 시스템을 형성하게 된다. 운용・보전



(그림 7) 코드 생성을 위한 시스템 구성도

기능 프로그램 개발에 이용되는 C++ 코드 생성을 위한 UML 작성 규칙을 살펴보면 아래와 같다.

◈ C++ 코드 생성을 위한 UML 작성 규칙

① Class diagram을 적당한 패키지로 분리하여 작성한다.
② cat file로 서로 다른 UML file의 class와의 관계를 설정한다.
③ C++ 생성 파일 디렉토리는 project property를 이용하여 기본 path를 설정한다.
④ UML로 만들어지지 않은 추가적 header file을 기술한다('Module Body'나 'Module Spec'에 기술).
⑤ Component Package를 생성, 헤더 및 C++ body가 생성될 디렉토리와 매핑한다.
⑥ Main Component Diagram에 Component Spec(header file)과 Component Body(C++body)를 생성하여 클래스를 각각 적당한 component에 할당한다(Component Spec/Body는 생성 파일명과 동일하다).
⑦ Componet Spec과 Componet Body 간 의존관계가 존재하면 Dependency 관계를 맺는다.
⑧ Component들을 해당 Component Package로 이동시킨다(생성될 디렉토리와 일치).
⑨ Component Package를 이용하여 코드를 생성한다.

IV. 실행 결과

II, III장에서 소개한 미들웨어와 UML을 이용하여 생성된 교환 소프트웨어 규모는 <표 6>과 같다.

<표 6> 소프트웨어 규모

구분	사이즈(LOC) 및 개수	카테고리
Netfree2000	idl/Makefile(50개), hdr(262개), lib(276개)	idl, hdr, lib, Makefile
UML	Mdl(19개), .h(85개)	*.mdl, *.h, *.cpp 생성
응용프로그램	480,189	소스(src)

*.cpp file은 생략(각 블록별로 수 개~수십 개가 생성됨)

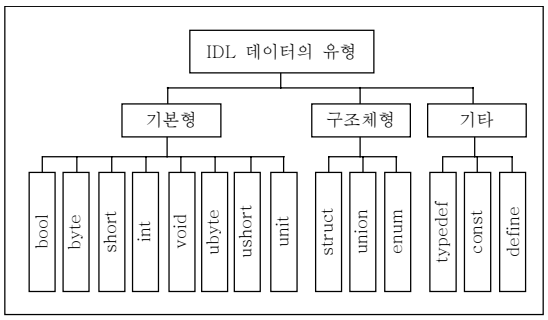
미들웨어(netfree2000)를 통해 생성되는 코드는 <표 6>에 언급한 것과 같이 헤더(hdr) 파일 및 라이브러리, 각종 시스템 공통파일(include file 포함)들을 링크시켜 하나의 실행모듈을 생산해 내는 Makefile, 소프트웨어 컴포넌트의 서비스를 프로그램언어에 독립적인 방식으로 구문적인 측면을 기술하는 IDL 파일 등이다. 즉, IDL 파일은 컴포넌트 개발 관점과 통합 관점을 기술하는 중계 역할을 한다. 이와 같은 컴포넌트 기술 방식은 각 컴포넌트를 조립하여 시스템을 구성하는 프로세스로 발전시켜 소프트웨어의 생산성 향상 및 유지보수 비용의 절감을 가져오게 되었다. 그러나 시스템 개발 초기에 소프트웨어 종합환경 설정에 대한 scope 확정 및 디렉토리 체계 확립, 관리 범위 등을 수립하는 데 많은 시행 착오를 거쳤다. 다시 말해서 소프트웨어 컴포넌트의 종류 및 유지보수성을 고려한 사후 관리 요령, 개발시스템과 운용시스템(워크스테이션) 간의 각종 컴포넌트와 관리 체계 등에 대한 결정에 많은 어려움이 수반되었다.

ACE2000 시스템 개발에 있어 소프트웨어 개발의 용이성과 관리의 통일성을 기하기 위해 개발시스템과 운용시스템의 환경을 일치시켜 사후 소프트웨어 이식(porting) 시 많은 오류를 줄이도록 한 것이 특징이다. IDL 작성 형식 및 데이터 유형은 아래와 같다((그림 8) 참조).

◈ IDL 작성 형식 예제

<서버명> {
  <서버의 유형 지정>;
  <라이브러리 서버의 Text Address 지정>;
  <라이브러리 서버의 Table Address 지정>;
  <참조하는 원격 서버명들>;
  <참조하는 라이브러리들>;
  <상수 정의>;
  <데이터 유형 정의>;
  <프로시저 또는 라이브러리 정의>;
  <전역변수 또는 공유메모리 정의>;
  <쓰레드 정의>;

<프로그램 정의>;
}
<서버명> {
}

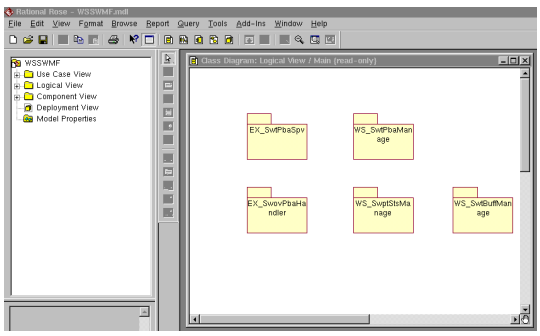


(그림 8) IDL 데이터 유형

UML을 사용해서 얻어진 코드는 주로 운용·보전기능의 프로그램으로 Rational사의 Rose를 사용, 모델링하여 얻어진 *.mdl file을 입력으로 logical view에서 얻어진 class diagram이 생성되면 이를 다시 component view에서 calss 당 하나의 컴포넌트를 생성하고 이를 그룹화하여 하나의 패키지화 한다. 이것들은 타 mdl과의 관계 등을 고려하고 블록간의 인터페이스 등을 정의한 코드를 삽입하여 툴에서 C++ file로 생성을 하면 최종 각 블록별 *.mdl file이 생성된다. 이때 생성되는 파일명은 기능블록명과 동일하게 함으로써 시스템 형상표에 의해 관리되는 소프트웨어 기능블록과 일치시킴으로써 개발자는 물론 관리자가 쉽게 파악할 수 있도록 하여 종합관리가 용이하도록 하였다. 예를 들면 AIM 상태관리를 위한 mdl file은 AIM2.mdl, 스위치 상태관리를 위한 워크스테이션 블록은 WSSWMF.mdl로, 시스템 시각관리를 위한 WSCLKMF.mdl 등으로 생성된다.

UML을 이용하여 생성된 mdl file 개수는 19개로 주로 워크스테이션 블록(10)과 성능(7), 과금(1) 및 정합모듈 상태관리 블록(1) 등이다. 이렇게 생성된 mdl 파일은 기능 연동을 위한 include file이 추가되어 conf file 및 Makefile 등과 함께 링크되어 컴파일되면 하나의 실행파일로 생성되어 교환기나 워크스테이션의 기능블록으로 로딩이 된다. 이것은 패키지 종합의 단위정보로 이용되는 software functional block(FB) 이다. 이 단위는 소프트웨어 관리 단위정보로도 이용된다.

(그림 9)는 UML 툴인 Rose를 실행하여 초기화면에서 스위치 상태관리 블록의 mdl file을 띄운 화면이다. 화면에 표시된 것과 같이 Rose의 디렉토리 구조는 use case view, logical view, component view, deployment view, model property로 구성되어 있다. 여기서 use case view는 요구사항으로부터 UML 기술을 명확하게 집중할 수 있는 view로써 사용자가 목표를 성취하기 위한 시스템과의 상호작용을 기술하는 것으로 전체 개발과정을 성립시키는 곳이다. 그 외에 deployment view에서는 소프트웨어 컴포넌트와 하드웨어 컴포넌트 사이의 physical 관계를 알아볼 수 있다. 즉, 컴포넌트와 오브젝트가 분산시스템에서 어떻게 라우트되고 이동하는가를 표시해준다. 또 model property는 UML 툴을 이용하여 생성할 코드에 대한 각종 환경 변수를 옵션으로 설정하도록 하여 리스크를 줄일 수 있도록 충분한 반복시험 기능을 제공한다[6],[7].



(그림 9) Rose 초기 화면(디렉토리 구조)과 Component Package 생성 예

ACE2000 시스템의 전체적인 소프트웨어 패키지 관리 디렉토리의 구조는 별첨의 디렉토리 계층구조를 참고하기 바란다[8].

디렉토리 구조는 크게 6개 부분으로 이에 대해 간단히 설명하면 우선 시스템 공통으로 사용되는 헤

더, 라이브러리 및 시스템에서 사용되는 각종 툴을 관리하는 공통부분과 소프트웨어 컴포넌트인 시그널, 모드 및 입·출력 메시지, 릴레이션과 도메인을 관리하는 디렉토리, 시스템의 형상관리를 위한 워크스테이션과 시스템 측의 형상데이터 파일, 각종 라이브러리 소스를 관리하는 디렉토리, 그리고 각 기능블럭들의 소스들을 관리하는 src와 미들웨어 관련 데이터들을 관리하는 netfree, 마지막으로 워크스테이션과 시스템에 로딩될 실행파일을 모아서 관리하는 EXEC 디렉토리로 나누어진다.

이 구조는 프로그램 개발용 시스템과 교환시스템 제어용 운용시스템인 워크스테이션과 동일하게 디렉토리를 구성하여 소프트웨어 관리자와 개발자, 시험자 및 프로젝트 참여 초보자도 개발 현황과 변경 여부를 쉽게 비교할 수 있도록 하였으며, 개발된 소프트웨어를 운용시스템으로 이식 시 파생될 수 있는 에러를 가능한 한 방지하도록 하였다.

V. 결론 및 향후 연구방향

본 논문에서는 ATM 스위칭 시스템을 개발하는데 있어서 새로운 개념의 개발방법을 도입, 실제 시스템에서 동작되어질 각 기능에 대한 구현사례로써 기존에 행해지던 개발방법을 탈피하고 미들웨어와 UML을 적용하여 시스템에 장착될 기능들에 대한 모델링과 이를 토대로 구현한 방법을 소개했다.

특히, 시스템의 성능 개선을 위해 워크스테이션과 교환기 간의 로딩될 블록들을 정의하고 디렉토리 체계를 확립하여 개발과정 상에 파생적으로 발생되는 에러들을 줄였으며, 주기적인 상태관리나 유지보수에 필요한 기능으로 실시간 처리가 덜 요구되는 기능들(주기적 보고나 상태관리, 로딩, 형상관리 등)을 시스템으로부터 워크스테이션으로 이동시킴으로써 시스템의 부하를 줄여 실시간 처리가 요구되는 호 제어 기능 및 자원관리 기능 등을 강화하였다.

또 시스템과 워크스테이션 간에 미들웨어를 도입함으로써 일관된 동작규격으로 정의된 기능을 수행하게 하므로 신속하게 에러의 원인 파악과 디버깅을 통해 시스템의 품질을 높일 수 있었다. 그러나 초기에는 신(新) 개념의 도입에 따른 경험부족과 툴에 대한 이해부족으로 인한 시행착오도 많이 발생했음을 본 고를 통해 밝혀둔다.

추후 고려사항으로는 이와 같은 새로운 개발 기법을 시스템 전체에 도입하여 시스템의 안정화와 양질의 소프트웨어를 개발해 내는 것이다. 금번 우리가 도입 적용한 분야는 주로 운용 보전기능과 호 제어기능의 일부에 적용하여 구현한 사례로 앞으로 좀더 많은 분야에 확대적용 및 개선이 필요한 부분이다.

참 고 문 헌

[1] 문미경, 염근혁, "도메인 엔지니어링 관련 연구 동향," 한국정보과학회 소프트웨어공학회지, 2001. 6., pp. 5 – 15.

[2] 권기태, "이질적인 소프트웨어 컴포넌트의 결합을 지원하는 형식 모델과 도구," 한국정보처리학회 소프트웨어 공학연구회지, 2000. 12., pp. 3 – 15.

[3] 박찬진, 우치수, "컴포넌트 정형명세," 한국정보처리학회지, Vol. 7, No. 4, 2000. 7., pp. 33 – 39.

[4] ㈜마하넷, "Netfree 2000 사용자 설명서 2.0," 2000. 10., pp. 3-1 – 3-19.

[5] 임재현, "UML C++ 코드 생성 방안," ㈜마하넷, 2000. 7., pp. 1 – 18.

[6] Grady Booch, James Rumbaugh, and Ivar Jacobson, "The Unified Modeling Language User Guide," Addison Wesley, 1999. 10., pp. 39 – 134.

[7] Martin Fowler and Kendall Scott, "UML Distilled Applying the Standard Object Modeling," 1997. 5., pp. 100 – 150.

[8] 신상권, "ACE2000 패키지 종합 매뉴얼," ETRI, 기술전수교재, 1999. 10.

<부록> 디렉토리 계층 구조(hostname: ace2k)

/pkg/idems /ACE2000/	Tool	HMI, DB, package 관련 각종 도구 모임			
	Syshdr	SROS, mSROS, dreams, tachyon header file			
	hdr	user common header file, shared library interface header file(NetFree 미사용)			
	Bin	PLD 제작도구 및 Makefile 생성도구, 기타 실행파일			
	syslib	SROS, mSROS, dreams, tachyon library archive			
	lib	user library archive, shared library interface archive(NetFree 미사용)			
	clshdr	UML에 의하여 생성되는 class header file			
	mdl	UML 디렉토리			
		tmp	UML에서 생성되는 불필요한 header directory		
	swc	sig	signal header file: 예, S7000_I_RteInfoReq.h		
		mode	typedef		
		rel	relation define		
		domain	domain define		
		process	process define		
		iomd	imd, omd, iomd_term		
			html	입출력 html 파일	
		hdr	Cxxxx.h, Sxxxx.h, Fxxxx.h, Axxxx.h, enumXXX.h		
	dg	국 데이터			
	dgpool	PLD 관련 디렉토리			
	wsdg	tachyon DBMS 초기 정보 생성			
	libsrc	LIBGSMP	1. library source, 2. shared library interface source(NetFree 미사용)		
		LIBHMI			
		…..			
		LIBCONF			
	src	UNPCF	SW block source		
		UNCCF			
		PCPF			
		WSFHF			
		WSCFGAF			
		WSCFGDF			
		…..			
	netfree	bin	name server		
		data	name table		
		syshdr	middleware 공통 header file		
		syslib	middleware run time library		
		hdr	idl compile 후 생성된 header file		
		lib	idl compile 후 생성된 archive file		
		idl	IDL file		
		client	MTP	c	C 언어 Client Makefile
				cpp	C++ 언어 Client Makefile
			…..		
	EXEC	cmd	WS command 처리 a.out		
		data	WS management data, hmi 메뉴 데이터		
		daemon	WS 실행 프로그램, JAVA JAR 프로그램, WS daemon program (WS 미들웨어 서버 포함)		
		bin	기타 실행파일		
		exec	교환기 Embedded processor 프로그램 및 데이터		
		DBMS	Tachyon 실행파일 및 wsdg		